We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

9

Particle Swarm Optimization and Other Metaheuristic Methods in Hybrid Flow Shop Scheduling Problem

M. Fikret Ercan
Singapore Polytechnic School of Electrical and Electronic Engineering
Singapore

1. Introduction

Multiprocessor task scheduling is a generalized form of classical machine scheduling where a task is processed by more than one processor. It is a challenging problem encountered in wide range of applications and it is vastly studied in the scheduling literature (see for instance (Chan & Lee, 1999 and Drozdowski, 1996) for a comprehensive introduction on this topic). However, Drozdowski (1996) shows that multiprocessor task scheduling is difficult to solve even in its simplest form. Hence, many heuristic algorithms are presented in literature to tackle multiprocessor task scheduling problem. Jin et al. (2008) present a performance study of such algorithms. However, most of these studies primarily concerned with a single stage setting of the processor environment. There are many practical problems where multiprocessor environment is a flow-shop that is it is made of multiple stages and tasks have to go through one stage to another.

Flow-shop scheduling problem is also vastly studied in scheduling context though most of these studies concerned with single processor at each stage (see for instance Linn & Zhang, 1999, Dauzère-Pérès & Paulli, 1997). With the advances made in technology, in many practical applications, we encounter parallel processors at each stage instead of single processors such as parallel computing, power system simulations, operating system design for parallel computers, traffic control in restricted areas, manufacturing and many others (see for instance (Krawczyk & Kubale, 1985, Lee & Cai, 1999, Ercan & Fung, 2000, Caraffa et. al.,2001)). This particular problem is defined as hybrid flow-show with multiprocessor tasks in scheduling terminology and minimizing the schedule length (makespan) is the typical scheduling problem addressed. However, Brucker & Kramer (1995) show that multiprocessor flow-shop problem to minimize makespan is also NP-hard. Gupta (1988) showed that hybrid flow-shop even with two stages is NP-hard. Furthermore, the complexity of the problem increases with the increasing number of stages.

Multiprocessor task scheduling in a hybrid flow-shop environment has recently gained the attention of the research community. To the best of our knowledge, one of the earliest papers that deal with this problem in the scheduling literature is by Oğuz and Ercan, 1997. However, due to the complexity of the problem, in the early studies (such as (Lee & Cai, 1999, Oğuz et. al., 2003)) researchers targeted two layer flow-shops with multiprocessors. Simple list based heuristics as well as meta-heuristics were introduced for the solution

(Oğuz et al.,2003, Jdrzejowicz & Jdrzejowicz,2003, Oğuz et al.,2004). Apparently, a broader form of the problem will have arbitrary number of stages in the flow-shop environment. This is also studied recently and typically metaheuristic algorithms applied to minimize the makespan such as population learning algorithm (Jdrzejowicz & Jdrzejowicz, 2003), tabu search (Oğuz et al.,2004), genetic algorithm (Oğuz & Ercan,2005) and ant colony system (Ying & Lin,2006). Minimizing the makespan is not the only scheduling problem tackled; recently Shiau et al. (2008) focused on minimizing the weighted completion time in proportional flow shops.

These metaheuristic algorithms produce impressive results though they are sophisticated and require laborious programming effort. However, of late particle swarm optimization (PSO) is gaining popularity within the research community due to its simplicity. The algorithm is applied to various scheduling problems with notable performance. For instance, Sivanandam et al. (2007) applied PSO to typical task allocation problem in multiprocessor scheduling. Chiang et al. (2006) and Tu et al. (2006) demonstrate application of PSO to well known job shop scheduling problem.

PSO, introduced by Kennedy & Eberhart (1995), is another evolutionary algorithm which mimics the behaviour of flying birds and their communication mechanism to solve optimization problems. It is based on a constructive cooperation between particles instead of survival of the fittest approach used in other evolutionary methods. PSO has many advantages therefore it is worth to study its performance for the scheduling problem presented here. The algorithm is simple, fast and very easy to code. It is not computationally intensive in terms of memory requirements and time. Furthermore, it has a few parameters to tune.

This chapter will present the hybrid flow-shop with multiprocessor tasks scheduling problem and particle swarm optimization algorithm proposed for the solution in details. It will also introduce other well known heuristics which are reported in literature for the solution of this problem. Finally, a performance comparison of these algorithms will be given.

2. Problem definition

The problem considered in this paper is formulated as follows: There is a set J of n independent and simultaneously available jobs where each job is made of Multi-Processor Tasks (*MPT*) to be processed in a multi-stage flow-shop environment, where stage j consists of m_j identical parallel processors ($j=1,2,...,k$). Each $MPT_i \in J$ should be processed on $p_{i,j}$ identical processors simultaneously at stage j without interruption for a period of $t_{i,j}$ ($i=1,2,...,n$ *and* $j=1,2,...,k$). Hence, each $MPT_i \in J$ is characterized by its processing time, $t_{i,j}$, and its processor requirement, $p_{i,j}$. The scheduling problem is basically finding a sequence of jobs that can be processed on the system in the shortest possible time. The following assumptions are made when modeling the problem:

- All the processors are continuously available from time 0 onwards.
- Each processor can handle no more than one task at a time.
- The processing time and the number of processors required at each stage are known in advance.
- Set-up times and inter-processor communication time are all included in the processing time and it is independent of the job sequence.

3. Algorithms

3.1 The basic PSO algorithm

PSO is initialized with a population of random solutions which is similar in all the evolutionary algorithms. Each individual solution flies in the problem space with a velocity which is adjusted depending on the experiences of the individual and the population. As mentioned earlier, PSO and its hybrids are gaining popularity in solving scheduling problems. A few of these works tackle the flow shop problem (Liu et al.,2005) though application to hybrid flow-shops with multiprocessor tasks is relatively new (Ercan & Fung, 2007, Tseng & Liao, 2008).

In this study, we first employed the global model of the PSO (Ercan & Fung, 2007). In the basic PSO algorithm, particle velocity and position are calculated as follows:

$$V_{id}=W\,V_{id} + C_1R_1(P_{id}-X_{id})+C_2R_2(P_{gd}-X_{id}) \quad (1)$$

$$X_{id}=X_{id}+V_{id} \quad (2)$$

In the above equations, V_{id} is the velocity of particle i and it represents the distance traveled from the current position. W is inertia weight. X_{id} represents particle position. P_{id} is the local best solution (also called as "pbest") and P_{gd} is global best solution (also called as "qutgbest"). C_1 and C_2 are acceleration constants which drive particles towards local and global best positions. R_1 and R_2 are two random numbers within the range of [0, 1]. This is the basic form of the PSO algorithm which follows the following steps:

Algorithm 1: The basic PSO

```
Initialize swarm with random positions and velocities;
begin
repeat
   For each particle evaluate the fitness i.e. makespan of the schedule;
   if current fitness of particle is better than Pid then set Pid to current value;
   if Pid is better than global best then set Pgd to current particle fitness value;
   Change the velocity and position of the particle;
until termination = True
end.
```

The initial swarm and particle velocity are generated randomly. A key issue is to establish a suitable way to encode a shedule (or solution) to PSO particle. We employed the method shown by Xia et al.(2006). Each particle consists of a sequence of job numbers representing the n number of jobs on a machine with k number of stages where each stage has m_j identical processors ($j=1,2,...,k$). The fitness of a particle is then measured with the maximum completion time of all jobs. In our earlier work (Oğuz & Ercan,2005), a list scheduling algorithm is developed to map a given job sequence to the machine environment and to compute the maximum completion time (makespan). A particle with the lowest completion time is a good solution.

Figure 1 shows an example to scheduling done by the list scheduling algorithm. In this example, number of jobs is n= 5 and a machine is made of two stages k=2 where each stage contains four identical processors. Table 1 depicts the list of jobs and their processing times and processor requirements at each stage for this example.

Job #	Stage 1 (j=1)		Stage 2 (j=2)	
i	$p_{i,1}$	$t_{i,1}$	$p_{i,2}$	$t_{i,2}$
1	1	1	2	2
2	3	3	4	2
3	3	3	3	2
4	2	1	2	1
5	1	1	1	1

Table 1. Example jobs and their processing time and processor requirements at each stage

For the schedule shown in Figure 1, it is assumed that a job sequence is given as S_1 ={2,3,1,4,5}. At stage 1, jobs are iteratively allocated to processors from the list starting from time 0 onwards. As job 2 is the first in the list, it is scheduled at time 0. It is important to note that although there are enough available processors to schedule job 1 at time 0 this will violate the precedence relationship established in the list. Therefore, job 1 is scheduled to time instance 3 together with job 3 and this does not violate the precedence relationship given in S_1 . Once all the jobs are scheduled at first stage, a new list is produced for the succeeding stage based on the completion of jobs at previous stage and the precedence relationships given in S_1. In the new list for stage 2, S_2 ={2,1,3,4,5} , job 1 is scheduled before job 3 since it is available earlier than job 3. At time instance 7, jobs 3, 4 and 5 are all available to be processed. Job 3 is scheduled first since its completion time is earlier at stage 1. Although, there is enough processor to schedule job 5 at time 8 this will again violate the order given in list S_1 , hence it is scheduled together with job 4. In this particular example, jobs 4 and 5 will be the last to be mapped to stage 2 and the over all completion time of tasks will be 10 units.

The parameters of PSO are set based on our empirical study as well as referring to the experiences of other researchers. The acceleration constants C_1 and C_2 are set to 2.0 and initial population of swarm is set to 100. Inertia weight, W, determines the search behavior of the algorithm. Large values for W facilitate searching new locations whereas small values provide a finer search in the current area. A balance can be established between global and local exploration by decreasing the inertia weight during the execution of the algorithm. This way PSO tends to have more global search ability at the beginning and more local search ability towards the end of the execution. In our PSO algorithm, an exponential function is used to set the inertia weight and it is defined as:

$$W = W_{end} + (W_{start} - W_{end})e^{-\frac{x\alpha}{x_{\max}}} \tag{3}$$

where, W_{start} is the starting, W_{end} is the ending inertia values. W_{start} are W_{end} are set as 1.5 and 0.3 respectively. In addition, x shows the current iteration number and x_{max} shows the maximum iteration number which is set to 10000. An integer constant α is used to manipulate the gradient of the exponentially decreasing W value and it is set to 4.

In this application, X_{id} and V_{id} are used to generate and modify solutions therefore they are rounded off to the nearest integer and limited to a maximum value of n which is the maximum number of jobs. That is position coordinates are translated into job sequence in our algorithm and a move in search space is obtained by modifying the job sequence.

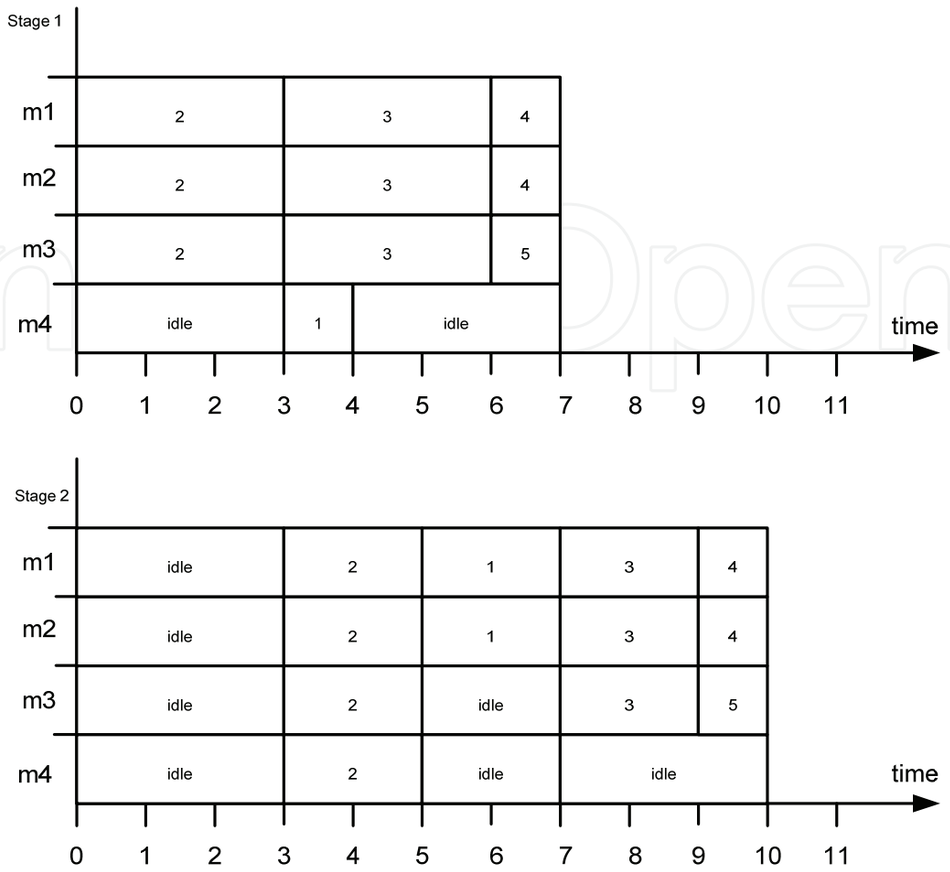


Figure 1. The schedule of job sequence [2, 3, 1, 4, 5] after being allocated to processors of the multilayer system using the list scheduling algorithm. Idle periods of the processors are labeled as idle

3.2 Hybrid PSO algorithms

Although, PSO is very robust and has a well global exploration capability, it has the tendency of being trapped in local minima and slow convergence. In order to improve its performance, many researchers experimented with hybrid PSO algorithms. Poli et al. (2007) give a review on the varations and the hybrids of particle swarm optimisation. Similarly, in scheduling problems, performance of PSO can be improved further by employing hybrid techniques. For instance, Xia & Wu (2006) applied PSO-simulated annealing (SA) hybrid to job shop scheduling problem and test its performance with benchmark problems. Authors conclude that PSO-SA hybrid delivered equal solution quality as compared to other metaheuristic algorithms though PSO-SA offered easier modeling, simplicity and ease of implementation. These findings motivated us to apply PSO and its hybrids to this particular scheduling problem and study its performance.

The basic idea of the hybrid algorithms presented here is simply based on runnign PSO algorithm first and then improving the result by employing a simulated annealing (SA) or tabu search (TS) heuristics. SA and TS introduce a probability to avoid becoming trapped in a local minimum. In addition, by introducing a neighborhood formation and tuning the

parameters, it is also possible to enhance the search process. The initial findings of this study are briefly presented in (Ercan,2008). The following pseudo codes show the hybrid algorithms:

Algorithm 2: Hybrid PSO with SA

```
Initialize swarm with random positions and velocities;
  begin
  initialize PSO and SA;
    while (termination !=true)
    do{
     generate swarm;
     compute and find best Pgd;
     }
  set particle that gives best Pgd as initial solution to SA;
     while (Tcurrent>Temp_end)
      do{
      generate neighborhood;
      evaluate and update best solution and temperature;
      }
  end.
```

Algorithm 3: Hybrid PSO with TS

```
Initialize swarm with random positions and velocities;
  begin
  initialize PSO and TS;
     while (termination !=true)
     do{
     generate swarm;
     compute and find best Pgd;
     }
  set particle that gives best Pgd as initial solution to TS;
     while (termination!=true)
      do{
      generate sub set of neighborhoods;
      evaluate and update the best solution;
      update the tabu list;
      }
   end.
```

The initial temperature for PSO-SA hybrid is estimated after 50 randomly permuted neighborhood solutions of the initial solution. A ratio of average increase in the cost to acceptance ratio is used as initial temperature. Temperature is decreased using a simple cooling strategy $T_{current} = \lambda T_{current} - 1$. The best value for lambda is experimentally found and set as 0.998. The end temperature is set to 0.01.

A neighbor of the current solution is obtained in various ways.

- Interchange neighborhood: Two randomly chosen jobs from the job list are exchanged.

- Simple switch neighborhood: It is a special case of interchange neighborhood where a randomly chosen job is exchanged with its predecessor.
- Shift neighborhood: A randomly selected job is removed from one position in the priority list and put it into another randomly chosen position.

It is experimentally found that interchange method performs the best amongst all three. The interchange strategy is also found to be the most effective one for generating the sub-neighborhoods for TS.
In the tabu list, a fixed number of last visited solutions are kept. Two methods for updating the tabu list are experimented; elimination of the farthest solution stored in the list, and removing the worst performing solution from the list. In PSO-TS hybrid removing the worst performing solution from the list method is used as it gave a slightly better result.

4. GA algorithm

Genetic algorithms, introduced by Holland (1975), have been widely applied to many scheduling problems in literature (see for instance job-shop environment (Della et al.,1995) and (Dorndorf & Pesch,1995), flow-shop environment (Murata et al., 1996)). Genetic algorithms are also employed in hybrid flow-shops with multiprocessor environment (Oğuz & Ercan, 2005). In this work, authors proposed a new crossover operator, NXO, to be used in the genetic algorithm and compare its performance with well-known PMX crossover. They employed two selection criteria in NXO to minimize the idle time of the processors. Firstly, NXO basically aims to keep the best characteristics of the parents in terms of the neighbouring jobs. That is if two jobs are adjacent to each other in both parents with good fitness values, then NXO tries to keep this structure in the offspring. If there is no such structure, then next criteria is employed in which NXO tries to choose the next job that will fit well in terms of the processor allocations. The results show that the genetic algorithm performs better in terms of the percentage deviation of the solution from the lower bound value when new crossover operator is used along with the insertion mutation. Some of the results from this study are included in this paper for comparison.

5. Other heuristic methods

The ant colony system (Dorigo, 1997) is another popular algorithm which is widely used in optimisation problems. Recently, Ying & Lin (2006) applied ant colony system (ACS) to hybrid flow-shops with multiprocessors tasks. Authors determine the jobs-permutation at the first stage, by ACS approach. Other stages are scheduled using an ordered list which is obtained by referring to completion times of jobs at the previous stage. Authors also apply the same procedure to the inverse problem to obtain the backward schedules. After that they employ a local search approach to improve the best schedule obtained in current iteration. Their computational results show that ACS has better performance compared to TS or GA though their algorithm is not any simpler than that of TS or GA.
Recently, Tseng & Liao (2008) tackled the problem by using particle swarm optimization. Their algorithm differs in terms of encoding scheme to construct a particle, the velocity equation and local search mechanism when compared to the basic PSO and the hybrid PSO algorithms presented here. Based on their published experimental results, PSO algorithm developed by Tseng & Liao (2008) performs well in this scheduling problem. Lately, Ying (2008) applied iterated greedy (IG) heuristic in search of a simpler and more efficient

solution. The IG heuristic also shows a notable performance as it's tailored to this particular problem.

6. Experimental results

The performance of all the meta-heuristics described above is tested using intensive computational experiments. Similarly, performance of the basic PSO and the hybrid PSO algorithms, in minimizing the overall completion time of all jobs, is also tested using the same computational experiments. The effects of various parameters such as number of jobs and processor configurations on the performance of the algorithm are also investigated. The results are presented in terms of Average Percentage Deviation (APD) of the solution from the lower bound which is expressed as:

$$APD = \frac{C_{\max} - LB}{LB} \times 100 \tag{3}$$

Here, C_{max} indicates the completion time of the jobs and LB indicates the lower bound calculated for the problem instance. The lower bounds used in this performance study were developed by Oğuz et al. (2004) and it is given with the following formula:

$$LB = \left\{ \max_{i \in M} \left\{ \min_{j \in J} \sum_{l=1}^{i-1} t_{l,j} \right\} + \frac{1}{m_i} \sum_{j \in J} p_{i,j} \times t_{i,j} + \min_{j \in J} \left\{ \sum_{l=i+1}^{l} t_{l,j} \right\} \right\} \tag{4}$$

In the above formula, M and J represent the set of stages and set of jobs consecutively. We used the benchmark data available at Oğuz's personal web-site (http://home.ku.edu.tr/~coguz/). Data set contains instances for two types of processor configurations:

(i) **Random processor:** In this problem set, the number of processors in each stage is randomly selected from a set of {1,..., 5}

(ii) **Fixed processor:** In this case identical number of processors assigned at each stage which is fixed to 5 processors.

For both configurations, a set of 10 problem instances is randomly produced for various number of jobs (n=5, 10, 20, 50, 100) and various number of stages (k=2, 5, 8). For each n and k value, the average APD is taken over 10 problem instances.

Table 2 and 3 presents the APD results obtained for the basic PSO and the hybrid PSO algorithms. Furthermore, we compare the results with genetic algorithm developed by Oğuz and Ercan (2005), tabu search by Oğuz et al. (2004), ant colony system developed by Ying and Lin (2006), iterated greedy algorithm (IG) by Ying (2008) and PSO developed by Tseng & Liao (2008). The performance of GA (Oğuz and Ercan,2005) is closely related to the control parameters and the cross over and mutation techniques used. Therefore, in Tables 2 and 3, we include the best results obtained from four different versions of GA reported. The performance comparison given in below tables is fair enough as most of the authors were employing the same problem set. Furthermore, all the algorithms use the same LB. However there are two exceptions. For the GA, authors use an improved version of the LB than the one given in equation 4. In addition, the PSO developed by Tseng & Liao (2008) is tested with different set of problems and with the same LB as in GA. However, these problems

also have the same characteristic in terms of number of stage, generation methods for processor and processing time requirements, etc.

From the presented results in Table 2 and 3, it can be observed that TS delivers reasonably good results only in two stage case; whereas GA demonstrates a competitive performance for small to medium size problems. For large number of jobs (such as n=50, 100) and large number of stages (k=8), GA did not outperform ACS, IG or PSO. When we compare ACS with TS and GA, we can observe that it outperforms TS and GA in most of the cases. For instance, it outperforms GA in 8 out of 12 problems in random processor case (Table 2). Among those, the performance improvement was more than %50 in six cases. On the other hand, IG gives a better result when compared to ACS in all most all the cases. The IG heuristic shows notable performance improvement for large problems (n=50 and n=100). For example, in n=100 and k=8 case, IG result is %71 better as compared to GA, %95 compared to TS and %7 compared to ACS.

The basic PSO algorithm presented here approximates to GA and ACS results though it did not show a significant performance improvement. PSO outperformed GA 4 in 12 problems for random processors and 1 in 12 problems for fixed processors. The best performance improvement was 54%. On the other hand, PSO-SA hybrid outperformed GA 7 and ACS 3 in 12 problems. In most of the cases, PSO-SA and PSO-TS outperformed the basic PSO algorithm. Amongst the two hybrids experimented here, PSO-SA gave the best results. The best result obtained with PSO-SA was in 50-jobs, 5-stages case, where the improvement was about 59% when compared to GA but this was still not better than ACS or IG. However, PSO developed by Tseng & Liao (2008) gives much more competitive results. Although thier results are for different set of problems, it can be seen that their algorithm performance improves when the problem size increases. Authors compared their algorithm with GA and ACS using the same set of data and reported that their PSO algorithm supersedes them, in particular for large problems.

From the results, it can also be observed that when the number of processors are fixed, that is m_j =5, the scheduling problem becomes more difficult to solve and APD results are relatively higher. This is evident in the given results of different metaheuristic algorithms as well as the basic PSO and the hybrid PSO algorithms presented here. In the fixed processor case, PSO-SA, which is the best performing algorithm among the three PSO algorithms, outperformed GA in 3 out of 12 problems and the best improvement achieved was %34. The performance of ACS is better for large problems though IG is dominant in most of the problems. For the fixed problem case, PSO algorithm developed by (Tseng & Liao, 2008) did not show an exceptional performance when compared to GA or ACS for smaller problems though for large problems (that is n=50 and 100) their PSO algorithm outperforms all.

The execution time of the algorithms is another indicator of the performance though it may not be a fair comparison as different processors and compilers used for each reported algorithm in literature. For instance, the basic PSO and the hybrid PSO algorithms presented here are implemented using Java language and run on a PC with 2GHz Intel Pentium processor (with 1024 MB memory). GA (Oğuz & Ercan, 2005) implemented with C++ and run on a PC with 2GHz Pentium 4 processor (with 256 MB memory), IG (Ying, 2008) with visual C#.net and PC with 1.5GHz CPU and ACS (Ying & Lin, 2006) with Visual C++ and PC with 1.5 GHz Pentium 4 CPU. However, for the sake of completeness we execute GA, the basic PSO and the hybrid PSO on the same computing platform using one easy (k=2, n=10) and one difficult problem (k=8, n=100) for the same termination criterion of 10000 iterations for all the algorithms. Results are reported in Table 4, which illustrates the speed performance of PSO. It can be seen

that PSO is approximately 48% to 35% faster than reported GA CPU timings. The fast execution time of PSO is also reported by Tseng and Liao (2008). However, in our case hybrid algorithms were as costly as GA due to computations in SA and TS steps.

k	n	TS (Oğuz et al. 2004)	GA (Oğuz & Ercan, 2005)	ACS (Ying & Lin, 2006)	IG (Ying, 2008)	Basic PSO	PSO-SA	PSO-TS	PSO* (Tseng & Liao, 2008)
2	10	3.0	1.60	1.60	1.49	2.7	1.7	2.1	2.8*
	20	2.88	0.80	1.92	1.87	2.88	1.12	1.92	5.40
	50	2.23	0.69	2.37	2.21	2.38	2.4	2.4	2.30
	100	9.07	0.35	0.91	0.89	1.82	0.82	1.1	1.62
5	10	29.42	11.89	9.51	8.73	10.33	9.78	10.4	10.45
	20	24.40	5.54	3.07	2.97	8.6	3.19	4.53	6.04
	50	10.51	5.11	1.51	1.49	3.31	2.06	2.98	1.44
	100	11.81	3.06	1.05	1.03	2.11	1.05	1.77	2.80
8	10	46.53	16.14	16.50	13.91	18.23	16.14	17.5	19.01
	20	42.47	7.98	6.77	5.83	12.03	6.69	7.03	5.76
	50	21.04	6.03	2.59	2.47	5.98	3.0	4.19	2.91
	100	21.50	4.12	1.33	1.23	8.78	2.11	5.22	1.53

Table 2. APD of the algorithms for 10 random instances. Random processors case (m_j ~[1,5])
(*) Different problem set

k	n	TS (Oğuz et al. 2004)	GA (Oğuz & Ercan, 2005)	ACS (Ying & Lin, 2006)	IG (Ying, 2008)	Basic PSO	PSO-SA	PSO-TS	PSO* (Tseng & Liao, 2008)
2	10	10.82	6.13	12.62	8.85	13.8	10.11	12.8	12.75*
	20	7.25	7.10	10.73	6.93	10.75	9.59	10.73	6.05
	50	5.80	3.34	8.17	5.66	10.32	7.02	8.82	5.69
	100	5.19	2.87	5.66	5.04	7.43	3.21	6.43	6.56
5	10	45.14	11.32	26.09	23.49	29.6	11.32	22.2	19.58
	20	35.13	10.78	15.11	12.64	19.4	10.77	16.5	12.33
	50	28.64	14.91	13.11	11.29	14.17	13.24	13.86	12.47
	100	26.49	11.02	12.45	10.53	12.45	12.45	12.45	11.49
8	10	77.21	25.98	25.14	22.17	30.81	25.83	25.83	33.92
	20	62.99	24.13	25.18	22.79	26.74	24.34	25.02	24.98
	50	54.25	21.87	22.23	20.71	27.01	23.07	25.11	19.41
	100	36.05	19.46	13.90	12.85	20.39	14.43	17.9	15.93

Table 3. APD of the algorithms for 10 random instances. Fixed processor case (m_j =5)
(*) Different problem set

k	n	GA	PSO	PSO-SA	PSO-TS
2	10	12.14	6.3	13.5	12.94
8	100	2109.9	1388.67	4029.1	3816.3

Table 4. Average CPU time (in seconds) of GA, TS and PSO. Processors setting is m_j=5

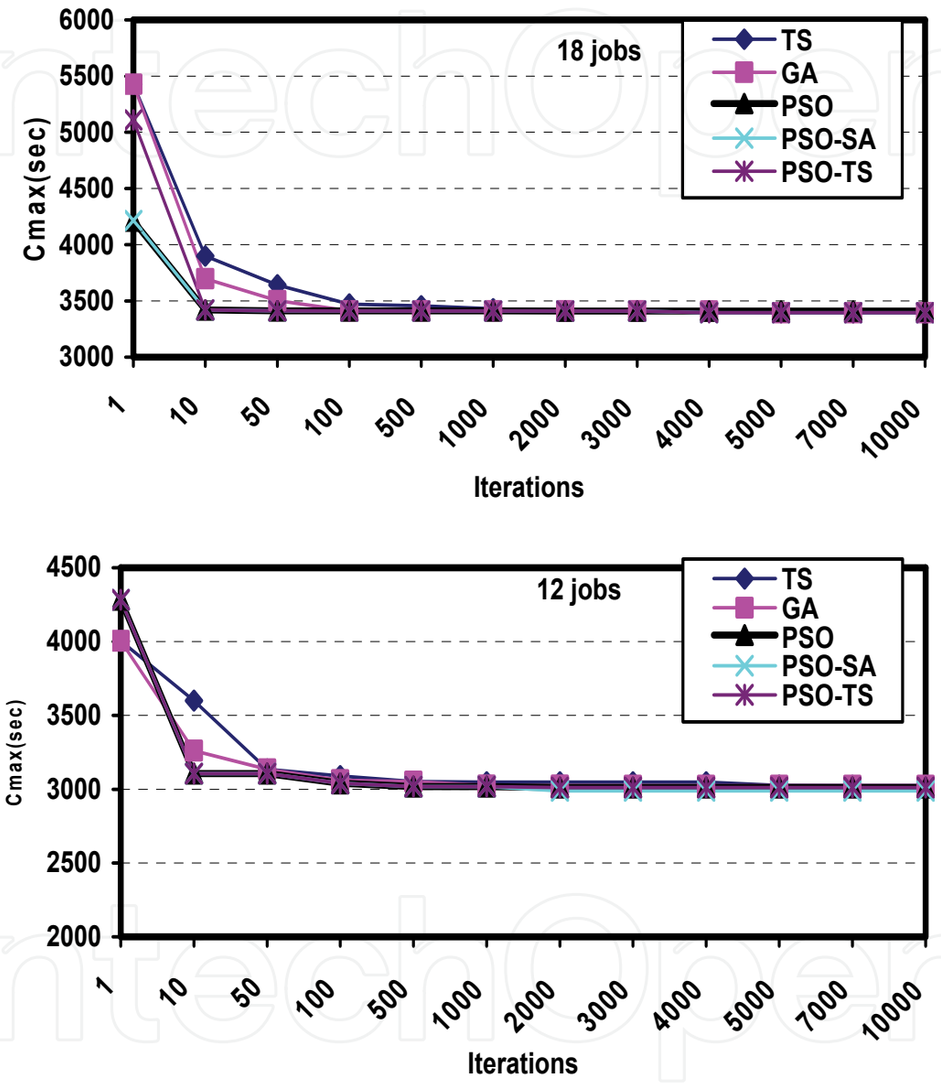


Figure 2. Convergence of TS, GA and PSO algorithms for a machine vision system

Lastly, we run the basic PSO and the hybrid PSO algorithms together with genetic algorithm and tabu search algorithm for the data obtained from a machine-vision system. The system is a multiprocessor architecture designed mainly for machine vision applications. The system comprise of two stages where each stage holds four identical DSP processors from Analog Devices. Data gathered from this system are for 8, 10, 12 and 18 jobs. The number of job is determined by the number of objects to be detected in a given image. The execution time and the processor requirements of parallel algorithms for each job are recorded in

order to use them as test problems in our scheduling algorithms. By utilizing this data, which can be obtained from the author, the convergence characteristic of the algorithms is analyzed. As it can be seen from Figure 2, all the algorithms converge very rapidly. However, PSO algorithms converge faster than TS and GA. In addition, PSO-SA finds a slightly better solution for 12 job problem. All the algorithms find a reasonably good solution within 1000 iterations. Hence, for practical application of the scheduling algorithms a small value for termination criteria can be selected.

7. Conclusion

In this chapter, a scheduling problem, defined as hybrid flow-shops with multiprocessor tasks, is presented together with various meta-heuristic algorithms reported for the solution in literature. As the solution to this scheduling problem has merits in practise, endeavour to find a good solution is worthy. The basic PSO and the hybrid PSO algorithms are employed to solve this scheduling problem, as PSO proven to be a simple and effective algorithm applied in various engineering problems. In this particular scheduling problem, a job is made up of interrelated multiprocessor tasks and each multiprocessor task is modelled with its processing requirement and processing time. The objective was to find a schedule in which completion time of all the tasks will be minimal. We observe that basic PSO has a competitive performance as compared to GA and ACS algorithms and superior performance when compared to TS. Considering the simplicity of the basic PSO algorithm, the performance achieved is in fact impressive. When experimented with the hybrids of PSO, it is observed that PSO-SA combination gave the best results. Hybrid methods improved the performance of PSO significantly though this is achieved at the expense of increased complexity. When compared to other published results on this problem, it can be concluded that IG algorithm (Ying, 2008) and PSO given by (Tseng & Liao, 2008) are the best performing algorithms on this problem so far. In terms of effort to develop an algorithm, execution time of algorithm and simplicity to tune it, PSO tops all the other metaheuristics.
As in many practical scheduling problems, it is likely to have precedence constraints among the jobs hence in future study hybrid flow-shops with precedence constraints will be investigated. In addition, PSO may be applied to other scheduling problems and its performance can be exploited in other engineering problems.

8. References

Brucker, P. & Kramer, B. (1995). Shop scheduling problems with multiprocessor tasks on dedicated processors, *Annals of Operations Research*, Vol. 50, 13-27

Caraffa, V.; Ianes, S.; Bagchi, T.P. & Sriskandarajah, C. (2001). Minimizing make-span in blocking flow-shop using genetic algorithms, *International Journal of Production Economics*, Vol. 70, 101-115

Chan, J. & Lee, C. Y. (1999). General multiprocessor task scheduling, *Naval Research Logistics,* Vol. 46, 57-74

Chiang, T. C.; Chang, P.Y. & Huang, Y. M. (2006). Multi-processor tasks with resource and timing constraints using particle swarm optimization, *International Journal of Computer Science and Network Security*, Vol.6, No.4, 71-77

Dauzère-Pérès, S. & Paulli, J. (1997). An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search, *Annals of Operations Research*, Vol. 70, 281-306

Dorigo, M. & Gambardella, L.M.(1997). Ant colony system: a cooperative learning approach to the travelling sales man problem. *IEEE Transaction in Evolutionary Computing*, Vol. 1, 53-66

Drozdowski, M. (1996). Scheduling multiprocessor tasks - an overview, *European Journal of Operational Research*, Vol. 94, 215-230

Ercan, M.F. & Fung, Y.F. (2000). The design and evaluation of a multiprocessor system for computer vision, *Microprocessors and Microsystems*, Vol. 24, 365-377

Ercan, M.F. and Fung, Y.F. (2007). Performance of particle swarm optimisation in scheduling hybrid flow-shops with multi-processor tasks, *Lecture Notes in Computer Science*, Vol. 4706, 309-319

Ercan M. F. (2008). A Performance Comparison of PSO and GA in Scheduling Hybrid Flow-Shops with Multiprocessor Tasks, *ACM Symposium on Applied Computing*, Ceara, Brasil.

Gupta J. N. D. (1988). Two stage hybrid flow shop scheduling problem. *Journal of Operational Research Society*, Vol. 39. No: 4, 359–364.

Holland J. H. (1975). *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor

Jdrzęjowicz, J. & Jdrzęjowicz, P. (2003). Population-based approach to multiprocessor task scheduling in multistage hybrid flow shops, *Lecture Notes in Computer Science*, Vol. 2773, 279-286

Jin, S.; Schiavone, G. & Turgut, D. (2008). A performance study of multiprocessor task scheduling algorithms, *Journal of Supercomputing*, Vol. 43, 77–97

Kennedy, J., & Eberhart R. (1995) Particle swarm optimization, *Proceedings of IEEE Int. Conf. on Neural Network*, pp. 1942-1948.

Krawczyk, H. & Kubale, M. (1985). An approximation algorithm for diagnostic test scheduling in multi-computer systems, *IEEE Trans. Computers*, Vol. 34/9, 869-8

Lee, C.Y. & Cai, X. (1999). Scheduling one and two-processors tasks on two parallel processors, *IIE Transactions*, Vol. 31, 445-455

Linn, R. & Zhang, W. (1999). Hybrid flow-shop schedule: a survey, *Computers and Industrial Engineering*, Vol. 37, 57-61 [9]

Liu, B.; Wang, L. & Jin, Y.H. (2005). Hybrid particle swarm optimization for flow shop scheduling with stochastic processing time, *Lecture Notes in Artificial Intelligence*, Vol. 3801, 630-637

Murata, T.; Ishibuchi, H. & Tanaka, H. (1996). Multi-objective genetic algorithm and its application to flow-shop scheduling, *Computers and Industrial Engineering*, Vol. 30, 957-968

Oğuz C. & Ercan M.F. (1997). Scheduling multiprocessor tasks in a two-stage flow-shop environment, *Computers and Industrial Engineering*, Vol. 33, 269-272

Oğuz, C.; Ercan, M.F.; Cheng, T.C.E. & Fung, Y.F. (2003). Heuristic algorithms for multiprocessor task scheduling in a two stage hybrid flow shop, *European Journal of Operations Research*, Vol.149, 390-403

Oğuz, C.; Zinder, Y.; Do., V.; Janiak, A. & Lichtenstein, M. (2004). Hybrid flow-shop scheduling problems with multiprocessor task systems, *European Journal of Operations Research*, Vol.152, 115-131

Oğuz, C. & Ercan, M. F. (2005). A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks, *Journal of Scheduling*, Vol. 8, 323-351

Poli, R.; Kennedy, J. & Blackwell, T. (2007). Particle swarm optimization an overview, *Swarm Intelligence*, Vol. 1, No. 3, 33-57.

Salman, A.; Ahmad, I. & Al-Madani, S. (2002). Particle swarm optimization for task assignment problem, *Microprocessors and Microsystems*, Vol. 26, 363-371

Shiau, D.F.; Cheng, S.C. & Huang, Y.M. (2008). Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm, *Expert Systems with Applications*, Vol. 34, 1133-1143

Sivanandam, S.N.; Visalakshi, P. and Bhuvaneswari, A. (2007). Multiprocessor scheduling using hybrid particle swarm optimization with dynamically varying inertia, *International Journal of Computer Science & Applications*, Vol. 4, 95-106

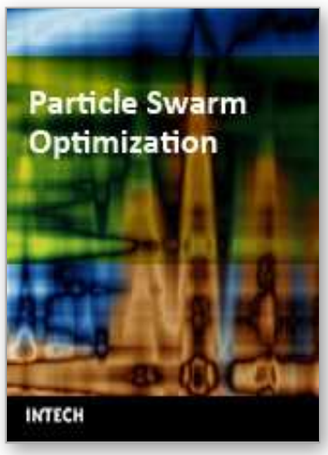
Tseng, C.T. & Liao, C.J. (2008), A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks, *International Journal of Production Research*, Vol. 46, 4655-4670.

Tu, K.; Hao, Z. & Chen, M. (2006). PSO with improved strategy and topology for job shop scheduling, *Lecture Notes in Computer Science*, Vol. 4222, 146-155

Xia, W.J. & Wu, Z.M. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem, *International Journal of Advance Manufacturing Technology*, Vol. 29, 360-366

Ying, K.C. & Lin, S.W. (2006). Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach, *International Journal of Production Research*, Vol. 44, 3161-3177

Ying, K.C. (2008). Iterated greedy heuristic for multiprocessor task scheduling problems, *Journal of the Operations Research Society (online edition)*, 1-8



Particle Swarm Optimization

Edited by Aleksandar Lazinica

ISBN 978-953-7619-48-0
Hard cover, 476 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Particle swarm optimization (PSO) is a population based stochastic optimization technique influenced by the social behavior of bird flocking or fish schooling.PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. This book represents the contributions of the top researchers in this field and will serve as a valuable tool for professionals in this interdisciplinary field.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen